Arayüz ve API kodunuzu Spyne ile Birleştirin

Burak Arslan
burak@arskom.com.tr

Özgür Web Teknolojileri Günleri

19 Ekim 2012

Konumuz

- Spyne'ı Kullanmak
  - Hello World
  - Çoklu Protokol
  - Doğrulama
  - SQLAlchemy entegrasyonu
  - İkincil Fonksiyonlar
- Katkıda bulunmak
  - Protokol yazmak
  - Taşıyıcı yazmak
  - İkincil işleyici yazmak
- XML Ekosisteminden neler öğrenebiliriz?

Spyne Nedir?

Spyne, birden fazla protokol ve/veya taşıyıcı kullanarak online hizmet sunmanızı kolaylaştırır.

Spyne Nedir?

Aynı zamanda uygulamanızın

iyi tanımlanmış bir online API

sunmasına yardımcı olur.

Nereden esti?

Nereden Esti?

- Eskiden;
  - Sadece tarayıcı sandbox'ına HTML/CSS/JS üretmek yeterliydi.

Nereden Esti?

- Eskiden;
  - Sadece tarayıcı sandbox'ına HTML/CSS/JS üretmek yeterliydi.
  - İstemci çok beceriksiz olduğu için grafik arayüz çizimi de sunucuda yapılıyordu.

Nereden Esti?

- Artık;
  - Tarayıcı dışındaki uygulama sandbox'larına ve sunulan hizmeti kullanan başka uygulamalara da veri üretiliyor.

Nereden Esti?

- Artık;
  - Tarayıcı dışındaki uygulama sandbox'larına ve sunulan hizmeti kullanan başka uygulamalara da veri üretiliyor.
  - İstemciler çok daha becerikli, o yüzden sunucunun üzerindeki bütün çizim yükü istemciye yüklenebiliyor.

Nasıl başladı?

Nasıl başladı?

- Önce protokol araştırmasına giriştim:
  - XML / SOAP / WSDL
  - XML / XML-RPC
  - JSON / JSON-RPC
  - Corba
  - Pickle
  - REST / HTTP
  - Vb

Neden SOAP?

XML ve XML Schema standartlarına dayanıyor.

XML Schema standardında:

- Teorik temelleri olan veri yapıları kadar (örn. 8/16/32/64 bit tamsayı tipleri) pratikte işe yarayan veri yapıları (örn. tarih/saat) da tanımlanmış.

Neden SOAP?

XML ve XML Schema standartlarına dayanıyor.

XML Schema standardında:

- Teorik temelleri olan veri yapıları kadar (örn. 8/16/32/64 bit tamsayı tipleri) pratikte işe yarayan veri yapıları (örn. tarih/saat) da tanımlanmış.

XML Standardının ise;

- Neredeyse her platformda desteği var.
- Doğrulayıcısı var (`lxml`).
- Sorgulama dili var (`XPath`).
- Gizlilik (confidentiality) ve bütünlük (integrity) desteği var (`PyXMLSec`).

Neden SOAP?

Soap, XML Schema standardına RPC, yönlendirme (routing) ve binary veri için iyileştirmeler (ve bir sürü başka özellik) ekliyor.

Dezavantajları?

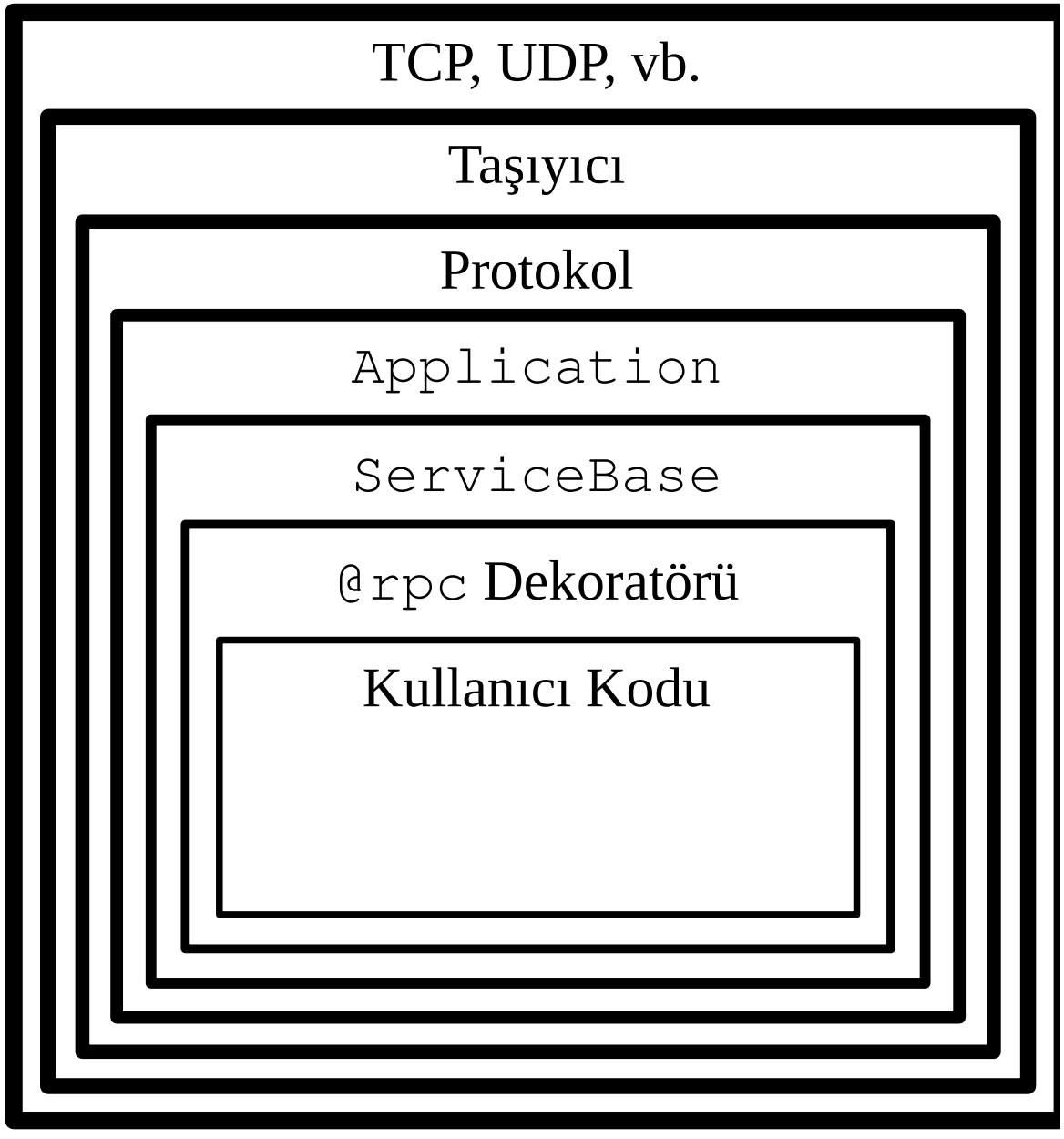
- **Yavaş!** Dokümanı hem işlemek yavaş hem de boyutu fazla.
- Soap standardının muğlak bıraktığı (dolayısıyla da uyumluluk sorunları çıkartan) bir çok nokta var.
  - Soap kötü ününün bir kısmını da bu belgeye borçludur.

Tamam, SOAP kullanmaya karar verdik...

Ama önce biraz terminoloji...

Terminoloji

- Taşıyıcı (Transport)
- Protokol
- Kullanıcı Kodu (User Code)


TCP, UDP, vb.
Taşıyıcı
Protokol
Application
ServiceBase
@rpc Dekoratörü
Kullanıcı Kodu

Soaplib ⇒ Rpclib ⇒ Spyne

2009	**Soaplib**	**0.8.1:** ~5000 İndirme
Kasım 2010		**1.0.0:** ~3000 İndirme
Mart 2011		**2.0.0-beta:** ~7000 İndirme
Mart 2012		**2.7.0-beta:** ~2000 İndirme

youtube.com/watch?v=N4zdWLuSbV0#t=21m58

Eylül 2012	**2.8.2-rc:** ~500 İndirme
29 Ekim 2012	**2.9.0:** 2 senedir çıkan ilk kararlı sürüm!

Soaplib ⇒ Rpclib ⇒ Spyne

2009		**0.8.1:** ~5000 İndirme
Kasım 2010	**Soaplib**	**1.0.0:** ~3000 İndirme
Mart 2011		**2.0.0-beta:** ~7000 İndirme
Mart 2012		**2.7.0-beta:** ~2000 İndirme

youtube.com/watch?v=N4zdWLuSbV0#t=21m58

Eylül 2012	**2.8.2-rc:** ~500 İndirme
29 Ekim 2012	**2.9.0:** 2 senedir çıkan ilk kararlı sürüm!

Soaplib ⇒ Rpclib ⇒ Spyne

2009		**0.8.1:** ~5000 İndirme
Kasım 2010		**1.0.0:** ~3000 İndirme
Mart 2011	**Soaplib**	**2.0.0-beta:** ~7000 İndirme
Mart 2012		**2.7.0-beta:** ~2000 İndirme

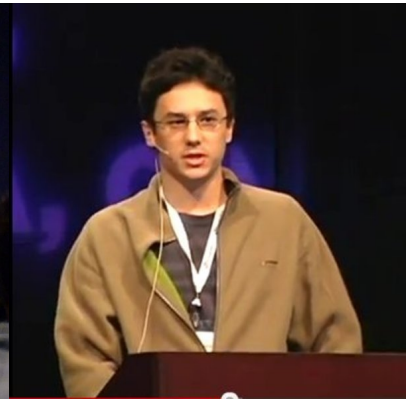
youtube.com/watch?v=N4zdWLuSbV0#t=21m58

Eylül 2012	**2.8.2-rc:** ~500 İndirme
29 Ekim 2012	**2.9.0:** 2 senedir çıkan ilk kararlı sürüm!

Soaplib ⇒ Rpclib ⇒ Spyne

2009		**0.8.1:** ~5000 İndirme
Kasım 2010		**1.0.0:** ~3000 İndirme
Mart 2011		**2.0.0-beta:** ~7000 İndirme
Mart 2012	**Rpclib**	**2.7.0-beta:** ~2000 İndirme





youtube.com/watch?v=N4zdWLuSbV0#t=21m58

Eylül 2012	**2.8.2-rc:** ~500 İndirme
29 Ekim 2012	**2.9.0:** 2 senedir çıkan ilk kararlı sürüm!

Soaplib ⇒ Rpclib ⇒ Spyne

2009	**0.8.1:** ~5000 İndirme
Kasım 2010	**1.0.0:** ~3000 İndirme
Mart 2011	**2.0.0-beta:** ~7000 İndirme
Mart 2012	**2.7.0-beta:** ~2000 İndirme

youtube.com/watch?v=N4zdWLuSbV0#t=21m58

Eylül 2012	**Spyne**	**2.8.2-rc:** ~500 İndirme
29 Ekim 2012		**2.9.0:** 2 senedir çıkan ilk kararlı sürüm!

Soaplib ⇒ Rpclib ⇒ Spyne

2009		**0.8.1:** ~5000 İndirme
Kasım 2010		**1.0.0:** ~3000 İndirme
Mart 2011		**2.0.0-beta:** ~7000 İndirme
Mart 2012		**2.7.0-beta:** ~2000 İndirme

youtube.com/watch?v=N4zdWLuSbV0#t=21m58

Eylül 2012		**2.8.2-rc:** ~500 İndirme
29 Ekim 2012	**Spyne**	**2.9.0:** 2 senedir çıkan ilk kararlı sürüm!

Şu an desteklenen Protokoller

XML:	SOAP 1.1 / XmlObject
Json:	JsonObject
MsgPack:	MessagePackObject, MessagePackRpc
Html:	HtmlMicroFormat(Ç), HtmlColumnTable(Ç), HtmlRowTable(Ç)
Http:	HttpRpc(G)
Csv:	Csv(Ç) ⇐ baştan yazmak gerekli

(G): Sadece giriş protokolü
(Ç): Sadece çıkış protokolü

Şu an desteklenen Taşıyıcılar

Http Client: urllib2, Twisted

Http Server: WSGI, Twisted, Django, Pyramid

Null Server: Test amaçlı çağrı arayüzü.

ZeroMQ: REQ/REP Soket tipi.

Buraya kadar sorusu olan?

Tamam, biraz da kod görelim...

Aşağıdaki basit fonksiyona bakalım:

```
from datetime import datetime

def get_utc_time():
    return datetime.utcnow()
```

Şimdi bu fonksiyonu uzaktan çağrılabilir yapmak için;

Şimdi bu fonksiyonu uzaktan çağrılabilir yapmak için;

1)

`@rpc` dekoratörü ile fonksiyonun girdi ve çıktı tiplerini belirliyoruz.

```
def get_utc_time():
    return datetime.utcnow()
```

```python
from spyne.model.primitive import DateTime
from spyne.decorator import srpc


def get_utc_time():
    return datetime.utcnow()
```

```python
from spyne.model.primitive import DateTime
from spyne.decorator import srpc

@srpc(_returns=DateTime)
def get_utc_time():
    return datetime.utcnow()
```

2)

Bir `ServiceBase` altsınıfına koyuyoruz.

```
from spyne.model.primitive import DateTime
from spyne.decorator import srpc


    @srpc(_returns=DateTime)
    def get_utc_time():
        return datetime.utcnow()
```

```
from spyne.model.primitive import DateTime
from spyne.decorator import srpc

from spyne.service import ServiceBase

class DateTimeService(ServiceBase):
    @srpc(_returns=DateTime)
    def get_utc_time():
        return datetime.utcnow()
```

3)

Daha sonra, bu hizmet tanımını kullanarak bir `Application` sınıfı oluşturuyoruz

```
[DateTimeService],
```

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc

httprpc = Application(
        [DateTimeService],
```

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc

httprpc = Application(
        [DateTimeService],
        tns='spyne.examples.multiprot',
```

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc

httprpc = Application(
        [DateTimeService],
        tns='spyne.examples.multiprot',
        in_protocol=HttpRpc(),
        out_protocol=HttpRpc()
    )
```

4)

Son olarak, uygulamayı bir 'taşıyıcı'ya bağlıyoruz.

```
from spyne.server.wsgi import WsgiApplication

application = WsgiApplication(httprpc)
```

Buradaki `application` nesnesi herhangi

bir WSGI uyumlu HTTP sunucusuna verebileceğimiz

standart bir WSGI uygulaması oldu.

```
from spyne.server.wsgi import WsgiApplication

application = WsgiApplication(httprpc)
```

Buradaki `application` nesnesi herhangi

bir WSGI uyumlu HTTP sunucusuna verebileceğimiz

standart bir WSGI uygulaması oldu.

```
$ curl http://localhost:9910/get_utc_time
2012-03-09T17:38:11.997784
```

Peki, ya biz bu fonksiyonu başka bir protokol kullanarak sunmak istersek ?

Misal: SOAP

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc
from spyne.protocol.soap import Soap11

soap = Application([DateTimeService],
         tns='spyne.examples.multiprot',
         in_protocol=HttpRpc(),
         out_protocol=Soap11()
     )
```

Misal: SOAP

```
$ curl http://localhost:9910/get_utc_time \
                                    | tidy -xml -indent
```

```
<?xml version='1.0' encoding='utf-8'?>
<senv:Envelope xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
 xmlns:tns="spyne.examples.multiple_protocols"
 xmlns:plink="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
 xmlns:xop="http://www.w3.org/2004/08/xop/include"
 xmlns:senc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:s12env="http://www.w3.org/2003/05/soap-envelope/"
 xmlns:s12enc="http://www.w3.org/2003/05/soap-encoding/"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:senv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <senv:Body>
    <tns:get_utc_timeResponse>
      <tns:get_utc_timeResult>
        2012-03-06T17:43:30.894466
      </tns:get_utc_timeResult>
    </tns:get_utc_timeResponse>
  </senv:Body>
</senv:Envelope>
```

Veya, düz XML:

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc
from spyne.protocol.xml import XmlObject

xml = Application([DateTimeService],
          tns='spyne.examples.multiprot',
          in_protocol=HttpRpc(),
          out_protocol=XmlObject()
      )
```

Veya, düz XML:

```
$ curl http://localhost:9910/get_utc_time \
                                    | tidy -xml -indent

<?xml version='1.0' encoding='utf-8'?>
<ns0:get_utc_timeResponse
xmlns:ns0="spyne.examples.multiple_protocols">
  <ns0:get_utc_timeResult>
    2012-03-06T17:49:08.922501
  </ns0:get_utc_timeResult>
</ns0:get_utc_timeResponse>
```

Veya, HTML:

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc
from spyne.protocol.xml import HtmlMicroFormat

html = Application([DateTimeService],
          tns='spyne.examples.multiprot',
          in_protocol=HttpRpc(),
          out_protocol=HtmlMicroFormat()
      )
```

Veya, HTML:

```
$ curl http://localhost:9910/get_utc_time \
                                    | tidy -xml -indent

<div class="get_utc_timeResponse">
  <div class="get_utc_timeResult">
    2012-03-06T17:52:50.234246
  </div>
</div>
```

vesaire...

Buraya kadar sorusu olan?

Giriş verisini deklaratif kısıtlarla sınırlamak da çok basit.

Yani, bunu yapacağınıza:

```
def get_name_of_month(month):
    """Aldigi 1-12 arasindaki tamsayiyi
    ay ismine donusturur.
    """
    # gelen veri int olmayabilir
    value = int(month)

    # gelen veri yukaridaki kisitlara uymayabilir
    if not (1 <= value <= 12):
        raise ValueError(value)

    return datetime(2000,month,1).strftime("%B")
```

Bu kadarı yeterli:

```
class NameOfMonthService(ServiceBase):
  @srpc(Integer(ge=1,le=12), _returns=Unicode)
  def get_name_of_month(month):
    return datetime(2000,month,1).strftime("%B")
```

Doğrulama özelliğini açmayı da unutmazsak;

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc

rest = Application([NameOfMonthService],
        tns='spyne.examples.multiprot',
        in_protocol=HttpRpc(validator='soft'),
        out_protocol=HttpRpc()
    )
```

```
$ curl localhost:9912/get_name_of_month?month=3
March
```

```
$ curl localhost:9912/get_name_of_month?month=3
March
```

```
$ curl -D - localhost:9912/get_name_of_month?month=13
HTTP/1.0 400 Bad Request
Date: Sat, 10 Mar 2012 14:21:36 GMT
Server: WSGIServer/0.1 Python/2.7.2
Content-Length: 63
Content-Type: text/plain

Client.ValidationError

The string '13' could not be validated
```

Buraya kadar sorusu olan?

Veritabanı ile çalışmak

Python ve ilişkisel veritabanlarını birarada kullanacaksanız,

SQLAlchemy kullanın!

(Gerçekten, Spyne kullanıp kullanmamanızın bu (⇑) sözle pek alakası yok.)

1)

Kendi veritabanı nesnenizi oluşturmanız lazım.

Spyne & SQLAlchemy

```
from spyne.model.complex import TTableModel
MyTableModel = TTableModel()
```

veya hazır metadatanız varsa;

```
MyTableModel = TTableModel(metadata)
```

veya hazır sınıfınız varsa;

```
MyTableModel.Attributes.sqla_metadata = \
                                        metadata
```

2)

Bu nesneyi kullanarak tablolarınızı

oluşturmanız lazım

Spyne & SQLAlchemy

```
class User(MyTableModel):
    __tablename__ = "ornek_user"
    id = Integer64(primary_key=True)
    isim = Unicode(64)
    soyad = Unicode(64)
```

Burada;

```
# SQAlchemy Tablo Nesnesi
User.Attributes.sqla_table

# SQAlchemy Mapper Nesnesi
User.Attributes.sqla_mapper
```

Spyne & SQLAlchemy

Yani tabloyu yaratmak için:

```
metadata.create_all()
```

veya

```
MyTableModel.Attributes.sqla_metadata.create_all()
```

veya

```
User.Attributes.sqla_table.create()
```

3)

Artık bu nesneyi kullanan

hizmetler yazabilirsiniz.

Spyne'ın karmaşık tanımlarını da veritabanına kolayca yazabilirsiniz

Spyne'ın SQLAlchemy Eklentileri

Şu iki nesne tanımına bakalım:

```
class Perm(MyTableModel):
    __tablename__ = 'ornek_user_perm'
    id = Integer32(primary_key=True)
    app = Unicode(256)
    op = Unicode(256)

class User(MyTableModel):
    __tablename__ = "ornek_user"
    id = Integer64(primary_key=True)
    name = Unicode(128)
    perms = Array(Perm)
```

Buradaki `Array` yapısını tek bir kolondaki bir XML dokümanı olarak tutmak istersek:

```
from spyne.model.complex import xml

class User(MyTableModel):
    __tablename__ = "ornek_user"
    id = Integer64(primary_key=True)
    name = Unicode(128)
    perms = Array(Perm).store_as(xml(
                                no_ns=True, root_tag="p"))
```

```
session.add(User(name="plq",
                 perms=[Perm(op='x', app='y'])))
```

$\Rightarrow$

```
INSERT INTO ornek_user (name,perms) VALUES (
  'plq','<p><Perm><op>x</op><app>y</app></Perm>');
```

Spyne'ın SQLAlchemy Eklentileri

Kendi tablosunda tutmak da isteyebiliriz:

```
from spyne.model.complex import table

class User(MyTableModel):
    __tablename__ = "ornek_user"
    id = Integer64(primary_key=True)
    name = Unicode(128)
    perms = Array(Perm).store_as(table())
```

```
session.add(User(name="plq",
                 perms=[Perm(op='x', app='y')]))
```

$\Rightarrow$

```
INSERT INTO ornek_user (name,perms)
   VALUES ('plq');
INSERT INTO ornek_perm (op,app,ornek_user_id)
   VALUES ('x', 'y', currval('ornek_user_id_seq'));
```

Servis örneği

```
class UserManagerService(ServiceBase):
    @rpc(User, _returns=Integer)
    def add_user(ctx, user):
        ctx.udc.session.add(user)
        ctx.udc.session.flush()

        return user.user_id
```

Servis örneği

Bu servisin kabul edeceği JSON dokümanı örneği:

```
{'User': {
  'isim': 'Burak',
  'soyad': 'Arslan',
  'perms': [
    {'app': 'y', 'op': 'x'},
    {'app': 'B', 'op': 'A'}
  ],
}}
```

Buraya kadar sorusu olan?

İkincil Fonksiyonlar

Spyne bağlamında;

Bir çağrının cevabı istemciye döndürüldükten sonra çalıştırılan fonksiyonlara “Auxiliary Method” diyoruz.

Servis örneğini hatırlayalım

```
class UserManagerService(ServiceBase):
    @rpc(User, _returns=Integer)
    def add_user(ctx, user):
        ctx.udc.session.add(user)
        ctx.udc.session.flush()

        return user.user_id
```

İkincil fonksiyon örneği

```
from spyne.auxproc.thread import ThreadAuxProc

class UserManagerAuxService(ServiceBase):
    __auxproc__ = ThreadAuxProc()

    @rpc(User, _returns=Integer)
    def add_user(ctx, user):
        mail_gonder("Patron! Bir kullanicimiz "
                    "daha oldu!")
```

İkincil fonksiyonun kullanılması

İkincil fonksiyon, birincil fonksiyon sorunsuz dönüş yaptıktan sonra arka planda çağrılır.

```
Application([
    UserManagerService,
    UserManagerAuxService,
],
tns='spyne.examples.multiprot',
    in_protocol=HttpRpc(),
    out_protocol=Soap11(),
)
```

İkincil fonksiyonlar ile ilgili son sözler

- Bir fonksiyon aksi belirtilmedikçe birincildir.
- Tam bir tane birincil fonksiyon ve sıfır veya daha fazla ikincil fonksiyon olabilir.

Buraya kadar sorusu olan?

Spyne'ı kullanmayı az çok öğrendik.

Peki Spyne'a nasıl katkıda bulunabiliriz?

Spyne'a Nasıl Katkıda Bulunabilirsiniz?

1. Protokol yazarak
2. Taşıyıcı yazarak
3. İkincil işleyici yazarak
4. Belge / Tutorial yazarak, tercüme yaparak.

Spyne ile kendi protokollerinizi yazmak da çok kolay.

Spyne ile kendi protokollerinizi yazmak da çok kolay.

kesin öyledir ☺

Örneğin, baştaki tarih ve saat örneğinde

dönüş verisini akrep ve yelkovanı olan

bir saat üzerinde göstermeye çalışalım.

(fazla detaya girmeden tabii ki)

Bunun için `ProtocolBase` alt sınıfında `serialize` ve `create_out_string` fonksiyonlarını yazmamız lazım.

```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'
```

```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'

  def serialize(self, ctx, message):




  def create_out_string(self, ctx, charset=None):
```

```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'

  def serialize(self, ctx, message):

    d = ctx.out_object[0] # fonksiyondan donen
                          # 'datetime' nesnesi




  def create_out_string(self, ctx, charset=None):
```

```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'

  def serialize(self, ctx, message):

    d = ctx.out_object[0] # fonksiyondan donen
                          # 'datetime' nesnesi

    # (detaya girmiyoruz demistik)



  def create_out_string(self, ctx, charset=None):
```

```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'

  def serialize(self, ctx, message):

    d = ctx.out_object[0] # fonksiyondan donen
                          # 'datetime' nesnesi

    # (detaya girmiyoruz demistik)

    # clock lxml'in Element nesnesi olarak tutulan
    # bir svg dosyasi
    ctx.out_document = clock

  def create_out_string(self, ctx, charset=None):
```

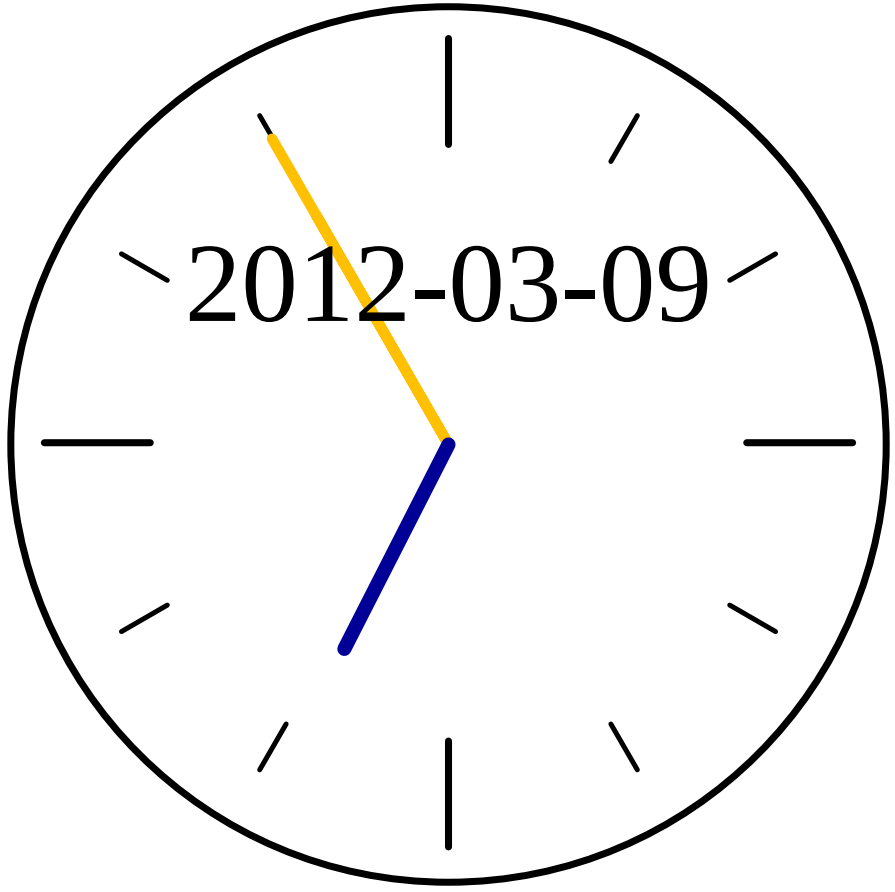
```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'

  def serialize(self, ctx, message):

    d = ctx.out_object[0] # fonksiyondan donen
                          # 'datetime' nesnesi

    # (detaya girmiyoruz demistik)

    # clock lxml'in Element nesnesi olarak tutulan
    # bir svg dosyasi
    ctx.out_document = clock

  def create_out_string(self, ctx, charset=None):

    ctx.out_string = [
        etree.tostring(ctx.out_document)
    ]
```

Bize özgü SVG protokolü:

```
svg = Application([DateTimeService],
        tns='spyne.examples.multiprot',
        in_protocol=HttpRpc(),
        out_protocol=SvgClock()
    )
```

Bize özgü SVG protokolü:

```
$ curl http://localhost:9910/get_utc_time \
                                    > utc_time.svg
```



Buraya kadar sorusu olan?

Spyne'da 10 çeşit taşıyıcı var:

1. İstemciler
10. Sunucular

İstemci taşıyıcısı

Aşağıdaki gibi bir modül yazılması gerekli:

```
class _RemoteProcedure(RemoteProcedureBase):
    def __call__(self, *args, **kwargs):
        self.ctx = self.contexts[0]

        self.get_out_object(self.ctx, args, kwargs)
        self.get_out_string(self.ctx)
        out_string = ''.join(self.ctx.out_string)

        # veriyi gonder

        # cevabi al

        self.ctx.in_string = in_string
        self.get_in_object(self.ctx)

        if not (self.ctx.in_error is None):
            raise self.ctx.in_error
        else:
            return self.ctx.in_object

class BenimSuperIstemcim(ClientBase):
    def __init__(self, url, app):
        ClientBase.__init__(self, url, app)

        self.service = Service(_RemoteProcedure, url, app)
```

İstemci taşıyıcısı

bunu okumanız gerekmiyordu ☺

İstemci taşıyıcısı yazmak

sunucu taşıyıcı yazmak ile

kıyaslandığında çok daha basittir.

Kaynak kod deposundaki örnekleri

incelemek çok daha faydalı olur.

Sunucu taşıyıcısı

Sunucu taşıyıcıları için belirlenmiş bir api yok.

Sunucu taşıyıcısı

Sunucu taşıyıcıları için belirlenmiş bir api yok.

Konu biraz karmaşık.

Sunucu taşıyıcısı

Yaptığı iş temelde şöyle:

- Giriş protokolüne istek bytestream'ini gönderip istek nesnesini al.

Sunucu taşıyıcısı

Yaptığı iş temelde şöyle:

- Giriş protokolüne istek bytestream'ini gönderip istek nesnesini al.
- Kullanıcı fonksiyonunu çalıştırıp dönen nesneyi al.

Sunucu taşıyıcısı

Yaptığı iş temelde şöyle:

- Giriş protokolüne istek bytestream'ini gönderip istek nesnesini al.
- Kullanıcı fonksiyonunu çalıştırıp dönen nesneyi al.
- Bu nesneden çıkış protokolüne bytestream ürettirip bunu istemciye gönder.

Sunucu taşıyıcısı

Bu işi yapan bir taşıyıcı yazarken:

- (Eğer varsa) Sonsuz döngü çağrısının adının `serve_forever()` olmasına,

Sunucu taşıyıcısı

Bu işi yapan bir taşıyıcı yazarken:

- (Eğer varsa) Sonsuz döngü çağrısının adının `serve_forever()` olmasına,
- Bütün concurrency yönetiminin burada yapılmasına,
  (Spyne kodunun geri kalanı *reentrant* olsa yeter.)

Sunucu taşıyıcısı

Bu işi yapan bir taşıyıcı yazarken:

- (Eğer varsa) Sonsuz döngü çağrısının adının `serve_forever()` olmasına,
- Bütün concurrency yönetiminin burada yapılmasına,
  (Spyne kodunun geri kalanı *reentrant* olsa yeter.)
- Hangi isteğin arayüz dokümanı için, hangisinin RPC için olduğuna karar verilen kısmı unutmamaya

Sunucu taşıyıcısı

Bu işi yapan bir taşıyıcı yazarken:

- (Eğer varsa) Sonsuz döngü çağrısının adının `serve_forever()` olmasına,
- Bütün concurrency yönetiminin burada yapılmasına,
  (Spyne kodunun geri kalanı *reentrant* olsa yeter.)
- Hangi isteğin arayüz dokümanı için, hangisinin RPC için olduğuna karar verilen kısmı unutmamaya

... dikkat ediyoruz.

Buraya kadar sorusu olan?

(itiraf etmek gerekirse bu bölümde yeteri kadar detaya girmedim)

İkincil işleyici yazma konusunda ise;

İkincil işleyici yazma konusunda ise;

apisi basit.

İkincil İşleyici Yazmak

(sadeleştirilmiş) ThreadAuxProc koduna bakalım:

```
class ThreadAuxProc(AuxProcBase):
  def process_context(self, server,
                      ctx, *args, **kwargs):



  def initialize(self, server):
```

İkincil İşleyici Yazmak

(sadeleştirilmiş) ThreadAuxProc koduna bakalım:

```
class ThreadAuxProc(AuxProcBase):
  def process_context(self, server,
                      ctx, *args, **kwargs):

    self.pool.apply_async(self.process,
                          (server, ctx) + args, kwargs)

  def initialize(self, server):

    self.pool = ThreadPool(self.__pool_size)
```

Bu konuda sorusu olan?

Peki, (şu veya bu sebeple) kendi mesajlaşma ekosistemini yaratmak isteyenler XML ve çevresindeki teknolojilerden neler öğrenebilir?

XML'i yakından inceledikten sonra gördük ki...

1. Namespace desteği bir çok şeyi kolaylaştırıyor.
2. Nesne tanımlama dokümanı (XML Schema) tanımı yapılmalı.
3. Binary veri etkin taşınmalı. (Soap'ın MTOM eklentisi)
4. (XMLSec) Kriptografik gizlilikte ve doğrulamada kullanılacak temel nesneler tanımlanmalı. Bu işlemleri (ve sıkıştırmayı da) "out-of-band" yapmamalı.
5. Kalıcı depolama formatı (EXI) ve sorgulama dili (XPath) tanımlamalı.
6. Mesaj yönlendirme ve RPC temel nesnelerini tanımlayın, API tanımlama standardı (WSDL) tanımlayın.
   - Yönlendirme verisini eklemek için bütün mesajı işlemek gerekmesin! Bkz. email.

Geçmiş Olsun!

umarım sizleri çok sıkmadım...

Eee, ne eksik?

Belge!

Belge!

Belge!

Belge!

Belge!

Belge!

Peki, başka ne eksik?

Protokollerden:	ProtoBuf! XmlRpc! Thrift! YAML! HTML! (Yani bütün sayfa)
Taşıyıcılardan:	SMTP! Bildiğin Dosya! SPDY! WebSockets!
İkincil İşleyicilerden:	Başka süreçte, hatta başka bilgisayarda işleme!

(ve daha bir sürü şey! Kaynak kod deposundaki ROADMAP.rst belgesine bir göz atabilirsiniz.)

Daha fazla bilgi için:

github.com/arskom/spyne

Örnekler

Çoklu protokol:	examples/multiple_protocols
Doğrulama:	examples/validation.py
Veritabanı:	examples/user_manager/server_sqlalchemy.py

Teşekkür

Bu sunuma değerli yorumlarıyla katkıda bulunan aşağıdaki arkadaşlarıma çok teşekkür ederim:

- Alper Aydemir
- Gökberk Arslan
- Merve Ünlü
- Abdurrahman Yaşar